Failure Detectors in Homonymous Distributed Systems (with an Application to Consensus)

Sergio ARÉVALO[⋆] Antonio FERNÁNDEZ ANTA[⋆⋆] Damien IMBS[‡]
Ernesto JIMÉNEZ[⋆] Michel RAYNAL[†,‡]

[⋆] EUI, Universidad Politécnica de Madrid, 28031 Madrid, Spain
[⋆⋆] Institute IMDEA Networks, 28918 Madrid, Spain
[†] Institut Universitaire de France
[‡] IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France

***Abstract*—This paper is on homonymous distributed systems where processes are prone to crash failures and have no initial knowledge of the system membership ("homonymous" means that several processes may have the same identifier). New classes of failure detectors suited to these systems are first defined. Among them, the classes $H\Omega$ and $H\Sigma$ are introduced that are the homonymous counterparts of the classes Ω and Σ, respectively. (Recall that the pair $\langle\Omega, \Sigma\rangle$ defines the weakest failure detector to solve consensus.) Then, the paper shows how $H\Omega$ and $H\Sigma$ can be implemented in homonymous systems without membership knowledge (under different synchrony requirements). Finally, two algorithms are presented that use these failure detectors to solve consensus in homonymous asynchronous systems where there is no initial knowledge of the membership. One algorithm solves consensus with $\langle H\Omega, H\Sigma\rangle$, while the other uses only $H\Omega$, but needs a majority of correct processes.**

Observe that the systems with unique identifiers and anonymous systems are extreme cases of homonymous systems from which follows that all these results also apply to these systems. Interestingly, the new failure detector class $H\Omega$ can be implemented with partial synchrony, while the analogous class $A\Omega$ defined for anonymous systems can not be implemented (even in synchronous systems). Hence, the paper provides us with the first proof showing that consensus can be solved in anonymous systems with only partial synchrony (and a majority of correct processes).

***Keywords*-Agreement problem, Asynchrony, Consensus, Distributed computability, failure detector, Homonymous system, Message-passing, Process crash.**

I. INTRODUCTION

Homonymous systems Distributed computing is on mastering uncertainty created by adversaries. The first adversary is of course the fact that the processes are geographically distributed which makes impossible to instantaneously obtain a global state of the system. An adversary can be static (e.g., synchrony or anonymity) or dynamic (e.g., asynchrony, mobility, etc.). The net effect of asynchrony and failures is the most studied pair of adversaries.

This paper is on agreement in crash-prone message-passing distributed systems. While this topic has been deeply investigated in the past in the context of asynchrony and process failures (e.g., [17], [19]), we additionally consider here that several processes can have the same identity, i.e., the additional static adversary that is *homonymy*. A motivation for homonymous processes in distributed systems can be found in [12] where, for example, users keep their privacy taking their domain as their identifier (the same identifier is then assigned to all the users of the same domain). Observe that homonymy is a generalization of two cases: (1) having unique identifiers and (2) having the same identifier for all the processes (anonymity), which are the two extremes of homonymy.

We also assume that the distributed system has to face another static adversary, which is the fact that, initially, each process only knows its own identity. We say that the system has to work *without initial knowledge of the membership*. This static adversary has been recently identified as of significant relevance in certain distributed contexts [16].

How to face adversaries It is well-known that lots of problems cannot be solved in presence of some adversaries (e.g., [1], [2], [14], [20]). When considering process crash failures, the *failure detector* approach introduced in [8], [9] (see [18] for an introductory presentation) has proved to be very attractive. It allows to enrich an otherwise too poor distributed system to solve a given problem P, in order to obtain a more powerful system in which P can be solved.

A failure detector is a distributed oracle that provides processes with additional information related to failed processes, and can consequently be used to enrich the computability power of asynchronous send/receive message-passing systems. According to the type (set of process identities, integers, etc.) and the quality of this information, several failure detector classes have been proposed. We refer the reader to [19] where classes of failure detectors suited to agreement and communication problems, corresponding failure detector-based algorithms, and additional behavioral assumptions that (when satisfied) allow these failure detectors to be implemented are presented. It is interesting to observe that none of the original failure detectors introduced in [9] can be implemented without initial knowledge of the membership [16].

Aim of the paper Agreement problems are central as soon as one wants to capture the essence of distributed computing. (If processes do not have to agree in one way or another, the problem we have to solve is not a distributed computing problem!) The aim of this paper is consequently to understand the type of information on failures that is needed when one has to solve an agreement problem in presence of asynchrony, process crashes, homonymy, and lack of initial knowledge of the membership. As consensus is the most central agreement problem we focus on it.

Related work As far as we know, consensus in anonymous networks has been addressed first in [3], [13] ([13] considers different synchrony assumptions while [3] considers systems enriched with failure detectors). Connectivity requirements for agreement in anonymous networks is addressed in [15].

To the best of our knowledge, up to now agreement in homonymous systems has been addressed only in [12] and [7]. In the former paper the authors consider that, among the n processes, up to t of them can commit Byzantine failures. The system is homonymous in the sense that there are ℓ, $1 \leq \ell \leq n$, different authenticated identities, each process has one identity, and several processes can share the same identity. It is shown in that paper that $\ell > 3t$ and $\ell > \frac{3t+n}{2}$ are necessary and sufficient conditions for solving consensus in synchronous systems and partially synchronous systems, respectively. The latter paper [7] mainly explores consensus in a shared memory system with anonymous processes, and bounds the complexity (namely, individual write and step complexities) of solving consensus with the aid of an anonymous leader elector $A\Omega$ (see below). They show that if the system is homonymous instead of purely anonymous these bounds can be improved.

The consensus problem in anonymous asynchronous crash-prone message-passing systems has been recently addressed in [3] (for the first time to our knowledge). In such systems, processes have no identity at all[1]. This paper introduces an anonymous counterpart[2] (denoted $\overline{AP}$ later in [4]) of the perfect failure detector P introduced in [9]. A failure detector of class $\overline{AP}$ returns an upper bound (that eventually becomes tight) of the current number of alive processes. The paper then shows that there is an inherent price associated with anonymous consensus, namely, while the lower bound on the number of rounds in a non-anonymous system enriched with P is $t+1$ (where t is the maximum number of faulty processes), it is $2t+1$ in an anonymous system enriched with $\overline{AP}$. The algorithm proposed assumes knowledge of the parameter t.

More general failure detectors suited to anonymous distributed systems are presented in [4]. Among other results, this paper introduces the anonymous counterpart $A\Sigma$ of the quorum failure detector class Σ [11] and the anonymous counterpart $A\Omega$ of the eventual leader failure detector class Ω [8]. It also presents the failure detector class AP which is the complement of $\overline{AP}$. An important result of [4] is the fact that relations linking failure detector classes are not the same in non-anonymous systems and anonymous systems. This is also the case if processes do not know the number n of processes in the system (unknown membership in anonymous systems). If n is unknown, the equivalence between AP and $\overline{AP}$, shown in [4], does not hold anymore.

Regarding implementability, it is stated in [4] that $A\Omega$ is not *realistic* (i.e., it can not be implemented in an anonymous synchronous system [10]). If the membership is unknown, it is not hard to show that AP is not realistic either, applying similar techniques as those in [16]. On the other hand, while $\overline{AP}$ can be implemented in an anonymous synchronous system, it is easy to show that it cannot be implemented in most partially synchronous systems (e.g., in particular, in those with all links eventually timely).

Contributions As mentioned, we explore the consensus problem in homonymous systems. Additional adversaries considered are asynchrony, process crashes, and lack of initial knowledge of the membership. We can summarize the main contributions of this paper as follows.

First, the paper defines new classes of failure detectors suited to homonymous systems. These classes, denoted $H\Omega$ and $H\Sigma$, are shown to be homonymous counterparts of Ω and Σ, respectively. The interest on the latter classes is motivated by the fact that $\langle \Sigma, \Omega \rangle$ is the weakest failure detector to solve consensus in crash prone asynchronous message-passing systems for any number of process failures [11]. The paper also investigates the relations linking $H\Sigma$, $A\Sigma$ and Σ, and shows that both $H\Omega$ and $H\Sigma$ can be obtained from $\overline{AP}$ in asynchronous anonymous systems. As a byproduct, we also introduce a new failure detector class denoted $\Diamond H\overline{P}$, that is the homonymous counterpart of $\Diamond\overline{P}$ (the complement of $\Diamond P$ [9]), which we consider of independent interest.

Then, the paper explores the implementability of these classes of failure detectors. It presents an implementation of $\Diamond H\overline{P}$ in homonymous message-passing systems with partially synchronous processes and eventually timely links. This algorithm does not require that the processes know the system membership. Since $H\Omega$ can be trivially implemented from $\Diamond H\overline{P}$ without communication, $H\Omega$ is realistic and can

[1] They must also execute the same program, because otherwise they could use the program (or a hash of it) as their identity. We consider that it is the same if processes have no identity or they have the same identity for all processes, since a process that lacks an identity can choose a default value (e.g., $\perp$) as its identifier.

[2] In this paper, when we say that a failure detector A is the *counterpart* of a failure detector B we mean that, in a classical asynchronous system (i.e., where each process has its own identity) enriched with a failure detector of class A, it is possible to design an algorithm that builds a failure detector of the class B and vice-versa by exchanging A and B. Said differently, A and B have the same computability power in a classical crash-prone asynchronous system.

also be implemented in a partially synchronous homonymous system without membership knowledge. The paper also presents an implementation of $H\Sigma$ in a synchronous homonymous message-passing system without membership knowledge.

Finally, the paper presents two consensus algorithms for asynchronous homonymous systems enriched with $H\Omega$. Both algorithms are derived from consensus algorithms for anonymous systems proposed in [6] and [4], respectively. The main challenge, and hence, the main contribution of our algorithms, is to modify the original algorithms that used $A\Omega$ to use $H\Omega$ instead. In the second algorithm, also the use of $A\Sigma$ has been replaced by the use of $H\Sigma$.

The first algorithm assumes that each process knows the value n and that a majority of processes is correct in all executions[3]. Since, as mentioned, $H\Omega$ can be implemented with partial synchrony, the combination of the algorithms presented (to implement $H\Omega$ and to solve consensus with $H\Omega$) form a distributed algorithm that solves consensus in any homonymous system with partially synchronous processes, eventually timely links, and a majority of correct processes. Applied to anonymous systems, this result relaxes the known conditions to solve consensus, since previous algorithms were based on unrealistic failure detectors ($A\Omega$) or failure detectors that require a larger degree of synchrony ($\overline{AP}$).

The second consensus algorithm presented works for any number of process crashes, and does not need to know n, but assumes that the system is enriched with the pair of failure detectors $\langle H\Sigma, H\Omega\rangle$. This algorithm, combined with the algorithms to implement $H\Sigma$ and $H\Omega$, shows that the consensus problem can be solved in *synchronous* homonymous systems subject to any number of crash failures without the initial knowledge neither of the parameter t nor of the membership. Applied to anonymous systems, this result relaxes the known conditions to solve consensus under any number of failures, since previous algorithms used unrealistic detectors ($A\Omega$) or required to know t or an upper bound on it.

This second consensus algorithms also forces us to restate the conjecture of which could be the weakest failure detector to solve consensus in asynchronous anonymous systems. The algorithm solves consensus in anonymous systems with a pair of detectors $\langle H\Sigma, H\Omega\rangle$, and we describe how it can be modified to solve consensus with a pair $\langle H\Sigma, A\Omega\rangle$. Additionally, as mentioned, it is shown here that $H\Sigma$ can be obtained from $A\Sigma$, and both $H\Sigma$ and $H\Omega$ can be obtained from $\overline{AP}$. The conjecture issued in [4] was that $\langle A\Sigma, A\Omega\rangle \oplus \overline{AP}$ [4] could be the weakest failure detector. Then, using the same algorithm described in [4] to combine the consensus algorithms for $\langle H\Sigma, A\Omega\rangle$ and $\langle H\Sigma, H\Omega\rangle$, the new candidate to be the weakest failure detector for consensus despite anonymity is now $\langle H\Sigma, A\Omega\rangle \oplus \langle H\Sigma, H\Omega\rangle$.

Roadmap The paper is made up of V sections. Section II presents the system model. Section III introduces failure detector classes suited to homonymous systems, and explores their relation with other classes and their implementability. Finally, Section V presents failure detector-based homonymous consensus algorithms.

II. System Model

Homonymous processes Let Π denote the set of processes with $|\Pi| = n$. We use $id(p)$ to denote the identity of process $p \in \Pi$. Different processes may have the same identity, i.e. $p \neq q \not\Rightarrow id(p) \neq id(q)$. Two processes with the same identity are said to be *homonymous.* Let $S \subseteq \Pi$ be any subset of processes. We define $I(S)$ as the *multiset* (sometimes also called *bag*) of process identities in S, $I(S) = \{id(p) : p \in S\}$. Let us remember that, differently from a set, an element of a multiset can appear more than once. Hence, as $I(S)$ may contain several times the same identity, we always have $|I(S)| = |S|$. The *multiplicity* (number of instances) of identity i in a multiset I is denoted $mult_I(i)$. When I is clear from the context we will use simply $mult(i)$. $P(I) \subseteq \Pi$ is used to denote the processes whose identity is in the multiset I, i.e., $P(I) = \{p : p \in \Pi \wedge id(p) \in I\}$. Every process $p \in \Pi$ knows its own identity $id(p)$. Unless otherwise stated, a process p does not know the system membership $I(\Pi)$, nor the system size n, nor any upper bound t on the number of faulty processes. Observe that the set Π is a formalization tool that is not known by the set of processes of the system.

Processes are asynchronous, unless otherwise stated. We assume that time advances at discrete steps. We assume a global clock whose values are the positive natural numbers, but processes cannot access it. Processes can fail by crashing, i.e., stop taking steps. A process that crashes in a run is said to be *faulty* and a process that is not faulty in a run is said to be *correct*. The set of correct processes is denoted by $Correct \subseteq \Pi$.

Communication The processes can invoke the primitive $broadcast(m)$ to send a message m to all processes of the system (including itself). This communication primitive is modeled in the following way. The network is assumed to have a directed link from process p to process q for each pair of processes $p, q \in \Pi$ (p does not need to be different from q). Then, $broadcast(m)$ invoked at process p sends one copy of message m along the link from p to q, for each $q \in \Pi$. Unless otherwise stated, links are asynchronous and reliable, i.e., links neither lose messages nor duplicate messages nor corrupt messages nor generate spurious messages. If a process crashes while broadcasting

[3] The knowledge of n can be replaced by the knowledge of a parameter α such that, $\alpha > n/2$ and, in all executions, at least α processes are correct.

[4] $\oplus$ represents a form of composition in which the resulting failure detector outputs $\perp$ for a finite time until it behaves at all processes as one -and the same- of the two detectors that are combined.

a message, the message is received by an arbitrary subset of processes.

Notation and time-related definitions The previous model is denoted $HAS[\emptyset]$ (Homonymous Asynchronous System). We use $HPS[\emptyset]$ to denote a homonymous system where processes are partially synchronous and links are eventually timely. A process is *partially synchronous* if the time to execute a step is bounded, but the bound is unknown. A link is *eventually timely* if there is an unknown global stabilization time (denoted GST) after which all messages sent across the link are delivered in a bounded δ time, where δ is unknown. Messages sent before GST can be lost or delivered after an arbitrary (but finite) time.

$AS[\emptyset]$ denotes the classical asynchronous system with unique identities and reliable channels. Finally, $AAS[\emptyset]$ denotes the Anonymous Asynchronous System model [4]. Observe that $AS[\emptyset]$ and $AAS[\emptyset]$ are special cases (actually extreme cases with respect to homonymy) of $HAS[\emptyset]$ (an anonymous system can be seen as a homonymous system where all processes have the same default identifier $\perp$).

III. Failure Detectors

In this section we define failure detectors previously proposed and the ones proposed here for homonymous systems. Then, relationships between these detectors are derived, and their implementability is explored.

Failure detectors for classical and anonymous systems We briefly describe here some failure detector previously proposed. We start with the classes that have been defined for $AS[\emptyset]$.

A failure detector of class Σ [11] provides each process $p \in \Pi$ with a variable $trusted_p$ which contains a set of process identifiers. The properties that are satisfied by these sets are [Liveness] $\forall p \in Correct, \exists \tau \in N : \forall \tau' \geq \tau, trusted_p^{\tau'} \subseteq I(Correct)$, and [Safety] $\forall p, q \in \Pi, \forall \tau, \tau' \in N, trusted_p^{\tau} \cap trusted_q^{\tau'} \neq \emptyset$.

A failure detector of class Ω [8] provides each process $p \in \Pi$ with a variable $leader_p$ such that [Election] eventually all these variables contain the same process identifier of a correct process.

The following failure detector classes have been defined for anonymous systems $AAS[\emptyset]$.

A failure detector of class $A\Omega$ [4] provides each process $p \in \Pi$ with a variable a_leader_p, such that [Election] there is a time after which, permanently, (1) there is a correct process whose Boolean variable is true, and (2) the Boolean variables of the other correct processes are false.

A failure detector of class $\overline{AP}$ [3] provides each process $p \in \Pi$ with a variable $anap_p$ such that, if $anap_p^{\tau}$ and $Correct^{\tau}$ denote the value of this variable and the number of alive processes at time τ, respectively, then [Safety] $\forall p \in \Pi, \forall \tau \in N, anap_p^{\tau} \geq |Correct^{\tau}|$, and [Liveness] $\exists \tau \in N, \forall p \in Correct, \forall \tau' \geq \tau, anap_p^{\tau'} = |Correct|$.

A failure detector of class $A\Sigma$ [4] provides each process $p \in \Pi$ with a variable a_sigma_p that contains a set of pairs of the form (x, y). The parameter x is a label provided by the failure detector, and y is an integer. Let us denote $a\_sigma_p^{\tau}$ the value of variable a_sigma_p at time τ. Let $S_A(x) = \{p \in \Pi \mid \exists \tau \in N : (x, -) \in a\_sigma_p^{\tau}\}$. Any failure detector of class $A\Sigma$ must satisfy the following properties:

- Validity. No set a_sigma_p ever contains simultaneously two pairs with the same label.
- Monotonicity. $\forall p \in \Pi, \forall \tau \in N : (((x, y) \in a\_sigma_p^{\tau}) \implies (\forall \tau' \geq \tau : \exists y' \leq y : (x, y') \in a\_sigma_p^{\tau'})$.
- Liveness. $\forall p \in Correct, \exists \tau \in N : \forall \tau' \geq \tau : \exists (x, y) \in a\_sigma_p^{\tau'} : (|S_A(x) \cap Correct| \geq y)$.
- Safety. $\forall p_1, p_2 \in \Pi, \forall \tau_1, \tau_2 \in N, \forall (x_1, y_1) \in a\_sigma_{p_1}^{\tau_1} : \forall (x_2, y_2) \in a\_sigma_{p_2}^{\tau_2} : \forall T_1 \subseteq S_A(x_1) : \forall T_2 \subseteq S_A(x_2) : ((|T_1| = y_1) \wedge (|T_2| = y_2)) \implies (T_1 \cap T_2 \neq \emptyset)$.

Failure detectors for homonymous systems Classical failures detectors output a set of processes' identifiers. Our failures detectors extend this output to a multiset of processes' identifiers, due to the homonymy nature of the system. The following are the new failure detectors proposed for homonymous systems.

A failure detector of class $\Diamond H\overline{P}$ eventually outputs forever the multiset with the identifiers of the correct processes. More formally, a failure detector of class $\Diamond H\overline{P}$ provides each process $p \in \Pi$ with a variable $h\_trusted_p$, such that [Liveness] $\forall p \in Correct, \exists \tau \in N : \forall \tau' \geq \tau, h\_trusted_p^{\tau'} = I(Correct)$. This failure detector $\Diamond H\overline{P}$ is the counterpart of $\Diamond\overline{P}$.

A failure detector of class $H\Omega$ eventually outputs the same identifier ℓ and number c at all processes, such that ℓ is the identifier of some correct process, and c is the number of correct processes that have this identifier ℓ. More formally, a failure detector of class $H\Omega$ provides each process $p \in \Pi$ with two variables h_leader_p and $h\_multiplicity_p$, such that [Election] $\exists \ell \in I(Correct), \exists \tau \in N : \forall \tau' \geq \tau, \forall p \in Correct,\ h\_leader_p^{\tau'} = \ell$, and $h\_multiplicity_p^{\tau'} = mult_{I(Correct)}(\ell)$.

Any correct process p such that $id(p) = \ell$ is called a *leader*. Note that this failure detector does not choose only one leader, like in Ω or in $A\Omega$, but a set of leaders with the same identifier. When all identifiers are different, the class $H\Omega$ is equivalent to Ω. Furthermore, a failure detector of class $H\Omega$ can be obtained from any detector D of class $\Diamond H\overline{P}$ without any communication (for instance, setting at each process p periodically h_leader_p to the smallest element in $D.h\_trusted_p$, and $h\_multiplicity_p \leftarrow mult_{D.h\_trusted_p}(h\_leader_p)$).

A failure detector of class $H\Sigma$ provides each process $p \in \Pi$ with two variables h_quora_p and h_labels_p, where h_quora_p is a set of pairs of the form (x, m) (x is a label, and m is a multiset such that $m \subseteq I(\Pi)$) and h_labels_p is a set of labels. Roughly speaking, each pair (x, m) determines a set of quora, and the set h_labels_p of a process p determines in which of these sets it participates. More formal, let us denote $h\_quora_p^\tau$ and $h\_labels_p^\tau$ the values of variables h_quora_p and h_labels_p at time τ, respectively. Let $S(x) = \{p \in \Pi \mid \exists \tau \in N : x \in h\_labels_p^\tau\}$. Any failure detector of class $H\Sigma$ must satisfy the following properties:

- Validity. No set h_quora_p ever contains simultaneously two pairs with the same label.
- Monotonicity. $\forall p \in \Pi, \forall \tau \in N, \forall \tau' \geq \tau$: (1) $h\_labels_p^\tau \subseteq h\_labels_p^{\tau'}$, and (2) $((x, m) \in h\_quora_p^\tau) \implies \exists m' \subseteq m : (x, m') \in h\_quora_p^{\tau'}$.
- Liveness. $\forall p \in Correct, \exists \tau \in N : \forall \tau' \geq \tau, \exists (x, m) \in h\_quora_p^{\tau'} : m \subseteq I(S(x) \cap Correct)$.
- Safety. $\forall p_1, p_2 \in \Pi, \forall \tau_1, \tau_2 \in N, \forall (x_1, m_1) \in h\_quora_{p_1}^{\tau_1} : \forall (x_2, m_2) \in h\_quora_{p_2}^{\tau_2} : \forall Q_1 \subseteq S(x_1), \forall Q_2 \subseteq S(x_2), (I(Q_1) = m_1 \wedge I(Q_2) = m_2) \implies (Q_1 \cap Q_2 \neq \emptyset)$.

Comparing $H\Sigma$ and $A\Sigma$, one can observe that $H\Sigma$ has pairs (x, m) in which m is a multiset of identifiers, while $A\Sigma$ uses pairs (x, y) in which y is an integer. However, a more important difference is that, in $H\Sigma$, each process has two variables. Then, the labels that a process p has in h_quora_p can be disconnected from those it has in h_labels_p. This allows for additional flexibility in $H\Sigma$.

Reductions between failure detectors In this section we claim that it can be shown, via reductions, the relation of the newly defined failure detector classes with the previously defined classes. We use the standard form of comparing the relative power of failure detector classes of [9]. A failure detector class X is *stronger* than class X' in system $Y[\emptyset]$ if there is an algorithm A that emulates the output of a failure detector of class X' in $Y[X]$ (i.e., system $Y[\emptyset]$ enhanced with a failure detector D of class X). We also say that X' can be obtained from X in $Y[\emptyset]$. Two classes are equivalent if this property can be shown in both directions.

We only present here the main results. The proofs and additional details can be found in the Appendix. The first result shows that, in classical systems with unique identifiers, Σ, $H\Sigma$, and $A\Sigma$ are equivalent.

Theorem 1. *Failure detector classes Σ, $H\Sigma$, and $A\Sigma$ are equivalent in $AS[\emptyset]$. Furthermore, the transformations between Σ and $H\Sigma$ do not require initial knowledge of the membership.*

In anonymous systems we have the following properties. Recall that an anonymous system is assumed to be a homonymous system in which every process has a default identifier $\perp$[5].

Theorem 2. *Class $H\Sigma$ can be obtained from class $A\Sigma$ in $AAS[\emptyset]$ without communication.*

Theorem 3. *Classes $\diamond H\overline{P}$ and $H\Sigma$ can be obtained from class $\overline{AP}$ in $AAS[\emptyset]$ without communication.*

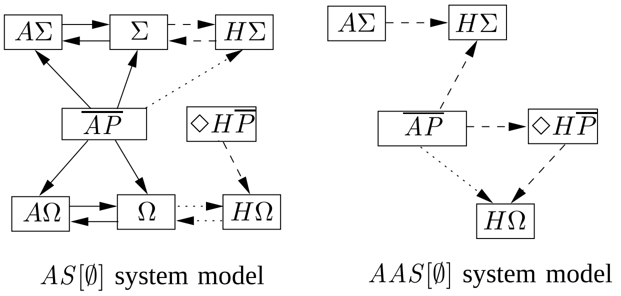

Figure 1. Relations between failure detector classes in the models $AS[\emptyset]$ and $AAS[\emptyset]$. There is an arrow from class X to X' if X is stronger that X'. Solid arrows are relations shown by Bonnet and Raynal in [4]. Dashed arrows are relations shown here, while dotted arrows are trivial relations.

IV. Implementing Failure Detectors in Homonymous Systems

In this section, we show that there are algorithms that implement the failure detectors classes $\diamond H\overline{P}$ and $H\Omega$ in $HPS[\emptyset]$ (homonymous partially synchronous system). We also implement the failure detector $H\Sigma$ in $HSS[\emptyset]$ (homonymous synchronous system). In all cases they do not need to know initially the membership.

A. Implementation of $\diamond H\overline{P}$ and $H\Omega$

The algorithm of Figure 2 implements $\diamond H\overline{P}$ (and $H\Omega$ with trivial changes) in $HPS[\emptyset]$ where processes are partially synchronous, links are eventually timely, and membership is not known.

Brief description of the algorithm: It is a polling-based algorithm that executes in rounds. At every round r, the Task 1 of each process p broadcasts $(POLLING, r, id(p))$ messages. After a time $timeout_p$, it gathers in the variable tmp_p (and, hence, also in $h\_trusted_p$) a multiset with the senders' identifiers id_s of processes from $(P\_REPLY, r', r'', id(p), id_s)$ messages received with $r' \leq r \leq r''$.

Task 2 is related with the reception of $POLLING$ and P_REPLY messages. When a process p receives a $(POLLING, r, id(q))$ message from process q, process p has to respond with as many P_REPLY as process q needs to receive up to round r, and not previously sent by process p (Lines 28-30). Note that the P_REPLY messages are piggybacked in only one message (Line 29). Also note that is in variable $latest\_r_p[id(q)]$ where p holds the latest round broadcast to $id(q)$. If it is the first time that process p receives a $(POLLING, -, id)$ message from a process

[5] Note that this differs from the assumption used in [4].

```
1  Init
2    $h\_trusted_p \leftarrow \emptyset$; // multiset of process identifiers
3    $mship_p \leftarrow \emptyset$; // set of process identifiers
4    $r_p \leftarrow 1$;
5    $timeout_p \leftarrow 1$;
6    start Tasks T1 and T2;
7
8  Task T1
9    repeat forever
10     broadcast $(POLLING, r_p, id(p))$;
11     wait $timeout_p$ time;
12     $tmp_p \leftarrow \emptyset$; // $tmp_p$ is an auxiliary multiset
13     for each $(P\_REPLY, r, r', id(p), id(q))$ received
14                  with $(r \leq r_p \leq r')$ do
15       add one instance of $id(q)$ to $tmp_p$;
16     end for;
17     $h\_trusted_p \leftarrow tmp_p$;
18     $r_p \leftarrow r_p + 1$;
19   end repeat;
20
21 Task T2
22   upon reception of $(POLLING, r_q, id(q))$ do
23     if $id(q) \notin mship_p$ then
24       $mship_p \leftarrow mship_p \cup \{id(q)\}$;
25       create $latest\_r_p[id(q)]$;
26       $latest\_r_p[id(q)] \leftarrow 0$;
27     end if;
28     if $latest\_r_p[id(q)] < r_q$ then
29       broadcast $(P\_REPLY, latest\_r_p[id(q)] + 1, r_q, id(q), id(p))$;
30     end if;
31     $latest\_r_p[id(q)] \leftarrow \max(latest\_r_p[id(q)], r_q)$;
32
33   upon reception of $(P\_REPLY, r, r', id(p), -)$ with $(r < r_p)$ do
34     $timeout_p \leftarrow timeout_p + 1$;
```

Figure 2. Algorithm that implements $\diamond H\overline{P}$ (code for process p).

with identifier id, then variable $latest\_r_p[id]$ is created and initialized to zero (Lines 23-27).

It is important to remark that, for each different identifier id, only one $(P\_REPLY, -, -, id(q), id)$ message is broadcast by each process q. So, if processes v and w with $id(v) = id(w) = x$ broadcast two $(POLLING, r, x)$ messages, then each process p only broadcast one (P_REPLY, r', r'', x, q) message with $r' \leq r \leq r''$. Note that eventually (at least after GST time) each P_REPLY message sent by any process has to be received by all correct processes. Hence, eventually processes v and w will receive all P_REPLY messages generated due to $POLLING$ messages.

Finally, Lines 33-34 of Task 2 allow process p to adapt the variable $timeout_p$ to the communication latency and process speed. When process p receives an outdated $(P\_REPLY, r, -, id(p), -)$ message (i.e., a message with round r less than current round r_p), then it increases its variable $timeout_p$.

Lemma 1. *Given processes $p \in Correct$ and $q \notin Correct$, there is a round r such that p does not receive any $(P\_REPLY, \rho, \rho', id(p), id(q))$ message from q with $\rho' \geq r$.*

Proof: There is a time τ at which q stops taking steps. If q ever sent a $(P\_REPLY, -, -, id(p), id(q))$ message, consider the largest x such that q sent message $(P\_REPLY, -, x, id(p), id(q))$. Otherwise, let $x = 0$. Then, the claim holds for $r = x + 1$. ■

Lemma 2. *Given processes $p, q \in Correct$, there is a round r such that, for all rounds $r' \geq r$, when p executes the loop of Lines 14-16 with $r_p = r'$, it has received a message $(P\_REPLY, \rho, \rho', id(p), id(q))$ from q with $\rho \leq r' \leq \rho'$.*

Proof: Observe that, since p is correct, it will repeat forever the loop of Lines 9-19, with the value of r_p increasing in one unit at each iteration. Hence, p will be sending forever messages $(POLLING, -, id(p))$ after GST with increasing round numbers, that will eventually be received by q. Then, q eventually will send infinite $(P\_REPLY, -, -, id(p), id(q))$ messages after GST, with increasing round numbers. Let $(P\_REPLY, x, -, id(p), id(q))$ be the first such message sent by q after GST. Then, for each round number $y \geq x$, there is some message $(P\_REPLY, \rho, \rho', id(p), id(q))$ sent by q with $\rho \leq y \leq \rho'$, and these messages are delivered at p at most δ time after being sent.

Now, assume for contradiction that for each round $y \geq x$, there is a round $y' \geq y$ such that, when p executes the loop of Lines 14-16 with $r_p = y'$, it has not received the message $(P\_REPLY, \rho, \rho', id(p), id(q))$ from q with $\rho \leq y' \leq \rho'$. But, every time this happens, when the message is finally received, r_p has been incremented in Line 18 and, hence, $timeout_p$ is incremented (in Lines 33-34). Then, eventually, by some round r, the value of $timeout_p$ will be greater than $2\delta + \gamma$, where γ is the maximum time that q takes to execute Lines 22-31. Then, p will receive message $(P\_REPLY, \rho, \rho', id(p), id(q))$ with $\rho \leq r' \leq \rho'$ before executing the loop of Lines 14-16 with $r_p = r'$, for all $r' \geq r$. We have reached a contradiction and the claim of the lemma follows. ■

Theorem 4. *The algorithm of Figure 2 implements a failure detector of the class $\diamond H\overline{P}$ in a system $HPS[\emptyset]$ (homonymous system where processes are partially synchronous and links are eventually timely), even if the membership is not known initially.*

Proof: Consider a correct process p. From Lemma 1, there is a round r such that p does not receive any $(P\_REPLY, \rho, \rho', -, -)$ message with $\rho' \geq r$ from any faulty process. From Lemma 2, there is a round r' such that for all rounds $r'' \geq r'$, when p executes the loop of Lines 14-16 with $r_p = r''$, it has received a $(P\_REPLY, \rho, \rho', -, -)$ message with $\rho \leq r'' \leq \rho'$ from each correct process. Hence, for every round $r'' \geq \max(r, r')$ when the Line 17 is executed with $r_p = r''$, the variable $h\_trusted_p$ is updated with the multiset $I(Correct)$. ■

We can obtain $H\Omega$ from the algorithm of Fig. 2 without additional communication. This can be done by simply including, immediately after Line 17, $h\_leader_p \leftarrow \min(h\_trusted_p)$ (i.e., the smallest identifier in $h\_trusted_p$) and $h\_multiplicity_p \leftarrow mult_{h\_trusted_p}(h\_leader_p)$.

Corollary 1. *The algorithm of Figure 2 can be changed to implement a failure detector of the class $H\Omega$ in a system $HPS[\emptyset]$ (homonymous system where processes are partially synchronous and links are eventually timely), even if the membership is not known initially.*

B. Implementation of $H\Sigma$

Figure 3 implements $H\Sigma$ in $HSS[\emptyset]]$ where processes are synchronous, links are timely, and membership is not known.

Brief description of the algorithm It runs in synchronous steps. In each step every process p broadcasts a $(IDENT, id(p))$ message. Then, process p waits for $(IDENT, -)$ messages sent through reliable links in this synchronous step by alive processes. Process p gathers in the multiset variable $mset_p$ the identifiers id of all $(IDENT, id)$ messages received. At the end of this step, variables h_quora_p and h_labels_p are updated with the value of $mset_p$. Note that for process p the label x of a quorum (x, m) is formed by the multiset $mset_p$ (i.e, $x = m = mset_p$).

Theorem 5. *The algorithm of Figure 3 implements a failure detector of the class $H\Sigma$ in a system $HSS[\emptyset]$ (homonymous synchronous systems), even if the membership is not known initially.*

Proof: From the definition of $H\Sigma$, it is enough to prove the following properties.

Validity. Since h_quora_p is a set, and the elements included in it are of the form $(mset, mset)$ (see Line 7 in Figure 3) there cannot be two pairs with the same label.

Monotonicity. The monotonicity of h_labels_p in Figure 3 holds because h_labels_p is initially empty, and each step, h_labels_p either grows or remains the same (see Line 8 in Figure 3). Similarly, the monotonicity of h_quora_p in Figure 3 follows from the fact that h_quora_p is initially empty, and any element $(mset, mset)$ included in it is never removed (see Line 7 in Figure 3).

Liveness. Let s be the synchronous step in which the last faulty process crashed. Then, in every step s' after s only correct processes will execute. Consider any process $p \in Correct$. In step s' will receive messages from all correct processes, and, hence, $mset_p = I(Correct)$. Then, process p includes $(I(Correct), I(Correct))$ in h_quora_p, and $I(Correct)$ in h_labels_p. Therefore, each correct process p is in $S(I(Correct))$. So, after step s, for each correct process p, the pair $(I(Correct), I(Correct))$ is in h_quora_p, and $I(Correct) = I(S(I(Correct)) \cap Correct)$.

```
1  $h\_labels_p \leftarrow \emptyset$;
2  $h\_quora_p \leftarrow \emptyset$;
3  for each synchronous step do
4      broadcast $(IDENT, id(p))$;
5      wait for the messages sent in this synchronous step;
6      $mset_p \leftarrow$ multiset of identifiers received in $(IDENT, -)$ messages;
7      $h\_quora_p \leftarrow h\_quora_p \cup \{(mset_p, mset_p)\}$
8      $h\_labels_p \leftarrow h\_labels_p \cup \{mset_p\}$;
9  end for;
```

Figure 3. Algorithm to implement $H\Sigma$ without knowledge of membership (code for process p)

Safety. Consider two pairs $(x_1, x_1) \in h\_quora_{p_1}^{\tau_1}$ and $(x_2, x_2) \in h\_quora_{p_2}^{\tau_2}$, for any $p_1, p_2 \in \Pi$ and any $\tau_1, \tau_2 \in N$.

Let M_1 be the set of processes from which p_1 received $(IDENT, -)$ messages in the synchronous step in which (x_1, x_1) was inserted for the first time in $h\_quora_{p_1}$. Observe that $Correct \subseteq M_1$. Furthermore, any process $p \in S(x_1)$ must also be in M_1 (i.e., $S(x_1) \subseteq M_1$). Also, $x_1 = I(M_1)$, and, hence, $|x_1| = |M_1|$. Therefore, the only set $Q_1 \subseteq S(x_1)$ such that $I(Q_1) = x_1$ is $Q_1 = M_1$. We define M_2 similarly, and conclude that the only set $Q_2 \subseteq S(x_2)$ such that $I(Q_2) = x_2$ is $Q_2 = M_2$. Since $Q_1 \cap Q_2 \supseteq Correct \neq \emptyset$, the safety property holds. ■

V. Solving Consensus in Homonymous Systems

We present in this section two algorithms. One algorithm implements Consensus in $HAS[t < n/2, H\Omega]$, that is, in an homonymous asynchronous system with reliable links, using the failure detector $H\Omega$, and when a majority of processes are correct. The other algorithm implements Consensus in $HAS[H\Omega, H\Sigma]$, that is, in an homonymous asynchronous system with reliable links, using the failure detector $H\Omega$ and $H\Sigma$.

A. Implementing Consensus in $HAS[t < n/2, H\Omega]$

Let us consider $HAS[t < n/2, H\Omega]$ where membership is unknown, but the number of processes is known (that is, n). Let us assume a majority of correct processes (i.e., $t < n/2$). We say that a process p is a leader, if it is correct and, after some finite time, $D.h\_leader_q = id(p)$ permanently for each correct process q. By definition of $H\Omega$, there has to be at least one leader.

The algorithm of Figure 4 is derived from the algorithm in Figure 4 of [6], proposed for anonymous systems. This algorithm has been adapted for homonymous systems. The algorithm of Figure 4 uses a failure detector of class $H\Omega$ (instead of $A\Omega$), and a new initial leaders' coordination phase has been added. The purpose of this initial phase is to guarantee that, after a given round, all leaders propose the same value in each round.

The algorithm works in rounds, and it has four phases (Leaders' Coordination Phase, Phase 0, Phase 1 and Phase

```
1  operation propose($v_p$):
2    $est1_p \leftarrow v_p$; $r_p \leftarrow 0$;
3    start Tasks T1 and T2;
4
5  Task T1
6    repeat forever
7      $r_p \leftarrow r_p + 1$;
8      // Leaders' Coordination Phase
9      broadcast $(COORD, id(p), r_p, est1_p)$;
10     wait until $(D.h\_leader_p \neq id(p)) \vee$
11       $(D.h\_multiplicity_p$ messages $(COORD, id(p), r_p, -)$ received);
12     if (some message $(COORD, id(p), r_p, -)$ received) then
13         $est1_p \leftarrow \min\{est_q : id(p) = id(q) \wedge$
14                        $(COORD, id(q), r_p, est_q)$ received $\}$ end if;
15     // Phase 0
16     wait until $(D.h\_leader_p = id(p) \vee ((PH0, r_p, v)$ received);
17     if $((PH0, r_p, v)$ received) then $est1_p \leftarrow v$ end if;
18     broadcast$(PH0, r_p, est1_p)$;
19     // Phase 1
20     broadcast$(PH1, r_p, est1_p)$;
21     wait until $(PH1, r_p, -)$ received from $n - t$ processes;
22     if (the same estimate $v$ received from $> n/2$ processes) then
23         $est2_p \leftarrow v$
24     else
25         $est2_p \leftarrow \perp$
26     end if;
27     // Phase 2
28     broadcast$(PH2, r_p, est2_p)$;
29     wait until $(PH2, r_p, -)$ received from $n - t$ processes;
30     let $rec_p = \{est2$ : message $(PH2, r_p, est2)$ received $\}$;
31     if $((rec_p = \{v\}) \wedge (v \neq \perp))$ then
32        broadcast $(DECIDE, v)$; return$(v)$ end if;
33     if $((rec_p = \{v, \perp\}) \wedge (v \neq \perp))$ then $est1_p \leftarrow v$ end if;
34     if $(rec_p = \{\perp\})$ then skip end if;
35   end repeat;
36
37 Task T2
38   upon reception of $(DECIDE, v)$ do
39     broadcast $(DECIDE, v)$; return$(v)$
```

Figure 4. Consensus algorithm in $HAS[t < n/2, H\Omega]$ (code for process p). It uses detector $D \in H\Omega$.

2). Every process p begins the Leaders' Coordination phase broadcasting a $(COORD, id(p), r, est1_p)$ message. If process p considers itself a leader (querying the failure detector D of class $H\Omega$), it has to wait until to receive $(COORD, id(p), r, est1)$ messages sent by all its homonymous processes (also querying the failure detector D of class $H\Omega$) (Lines 10-11). After that, process p updates its estimate $est1_p$ with the minimal value proposed among all its homonymous. Note that eventually all its homonymous will be leaders too. Hence, eventually all leaders will also choose the same minimal value in $est1$.

In Phase 0, if process p considers itself a leader (querying the failure detector D of class $H\Omega$) (Line 16), it broadcast a $(PH0, r, est1_p)$ message with its estimate in $est1_p$. Otherwise, process p has to update its $est1_p$ waiting until a $(PH0, r, est1_l)$ message is received from one of the leaders processes l (Lines 16-17). Note that after the Leaders' Coordination Phase, eventually each leader l broadcast $(PH0, -, est1_l)$ messages with the same value in $est1_l$.

The rest of the algorithm is similar to the algorithm in Figure 4 of [6]. We omit further details due to space restrictions. The following lemmas are the key of the correctness of the algorithm. They show that, even having multiple leaders, these will eventually converge to propose the same value at each round.

Lemma 3. *No correct process blocks forever in the Leaders' Coordination Phase.*

Proof: The only line in which processes can block in Lines 7-14 is in Lines 10-11. A correct process that is not leader does not block permanently in these lines, because eventually the first part of the wait condition is satisfied. Let us assume, for contradiction, that some leader blocks permanently in Line 11. Let us consider the smallest round r in which some leader p blocks. By definition of r, each leader q eventually reaches round r, and (even if it blocks in r) broadcasts $(COORD, id(q), r, -)$, where $id(q) = id(p)$, in Line 9. (Observe that all processes send $(COORD, -, -, -)$ messages in Line 9, even if they do not consider themselves as leaders.) Eventually, all these messages are delivered to p and $D.h\_multiplicity_p$ is permanently the number of leaders. Hence, the second part of the wait condition (Line 11) is satisfied. Thus, p is not blocked anymore, and, therefore, we reach a contradiction. ■

Lemma 4. *There is a round r such that at every round $r' > r$ all leaders broadcast the same value in Phase 0 of round r'.*

Proof: Eventually all leaders broadcast the same value because after some round, all leaders start Phase 0 with the same value in $est1$. Consider a time τ when all faulty processes have crashed and the failure detector D is stable (i.e., $\forall \tau' \geq \tau, \forall p \in Correct$, $D.h\_leader_p^{\tau'} = \ell$, being $\ell \in I(Correct)$, and $D.h\_multiplicity_p^{\tau'} = mult_{I(C)}(\ell)$). Let r be the largest round reached by any process at time τ. Then, for any round $r' > r$, all leaders p have the same estimate $est1_p$ at the beginning of the Phase 0 of round r' (Line 16), or there has been a decision in a round smaller than r'. To prove this, let us assume that no decision is reached in a round smaller than r'. Then, since the leaders do not block forever in any round (see previous paragraph 1), they execute Line 9 in round r'. Since the failure detector is stable, they also wait for the second part of the wait condition of Lines 10-11 (since the first part is not satisfied). When any leader p executes the Leaders' Coordination Phase of r', it blocks in Lines 10-11 until it receives $D.h\_multiplicity_p$ messages from the other leaders. By the stability of the $H\Omega$ failure detector, $D.h\_multiplicity_p$ is the exact number of leaders. Also, from the definition of τ and r, no faulty process with identifier $D.h\_leader_p$ is alive and all the messages they sent correspond to rounds smaller than r'. Hence, each leader p will wait to receive messages from all the other leaders and will set $est1_p$ to the minimum from the same set of values (Line 14). ■

Theorem 6. *The algorithm of Figure 4 solves consensus in $HAS[t < n/2, H\Omega]$.*

Proof: From the definition of Consensus, it is enough to prove the following properties.

Validity. The variable $est1$ is initialized with a value proposed by its process (Line 2). The value of $est1$ may be updated in Lines 14 or 17 with values of $est1$ broadcasted by other processes. The variable $est2$ is initialized and updated with $est1$ (Line 23) or $\perp$ (Line 25). The value of $est1$ may be updated in Line 33 with values of $est2$ (different from $\perp$) broadcasted by other processes. The value decided in Line 32 is the value of $est2$ that was broadcasted by some process. As it is not possible to decide the value $\perp$ (Line 32), then the value decided has to be one of the values proposed by the processes. Then, the validity property holds.

Agreement. Identical to the agreement property of Figure 4 of [6],

Termination. From Lemmas 3 and 4, after some round r, all leaders hold the same value v in $est1$ when they start executing Phase 0 of round r' (Line 16), and they broadcast this same value v (Line 18). Note that it is the same situation as having only one leader with value v stored in $est1$ when Phase 0 is reached. Hence, as Phase 0 starts in the same conditions as in the algorithm of Figure 4 of [6], the same proof can be used to prove the termination property. ■

B. Implementing Consensus in $HAS[H\Omega, H\Sigma]$

Figure 5 implements Consensus in $HAS[H\Omega, H\Sigma]$. Note that it is a variation of the algorithm of Figure 3 of [4] where, like in the previous case, we have added a preliminary phase as a barrier such that homonymous leaders eventually "agree" in the same estimation value $est1$ to propose. Once this issue has been solved (as was proven for the previous algorithm), the use that this algorithm makes of the failure detector $H\Sigma$ is very similar to the use the algorithm of Figure 3 of [4] makes of the $A\Sigma$ failure detector.

Lemma 5. *No correct process blocks forever in the repeat loops of Phases 1 and 2.*

Proof: Note that if a correct process decides (Line 51), then the claims follows. Consider the repeat loop of Phase 1 (Lines 22-38). Let us assume that some correct process is blocked forever in this loop. Then, let us consider the first round r in which a correct process blocks forever in r. Hence, all correct processes must block forever in the same loop in round r. Otherwise some process broadcasts a message $(PH2, -, r, -, -, -)$, and from Line 24 no correct process would block forever in this loop of round r. Let us consider a correct process p, and the pair (x, m) that

```
1  operation propose($v_p$):
2    $est1_p \leftarrow v_p$; $r_p \leftarrow 0$;
3    **start** Tasks T1 **and** T2;
4
5    **Task** T1
6     **repeat forever**
7      $r_p \leftarrow r_p + 1$;
8      // Leaders' Coordination Phase
9      broadcast $(COORD, id(p), r_p, est1_p)$;
10     **wait until** $(D1.h\_leader_p \neq id(p)) \vee$
11        $(D1.h\_multiplicity_p$ messages $(COORD, id(p), r_p, -)$ received);
12     **if** (some message $(COORD, id(p), r_p, -)$ received) **then**
13        $est1_p \leftarrow \min\{est_q : id(p) = id(q) \wedge$
14                        $(COORD, id(q), r_p, est_q)$ received $\}$ **end if**;
15     // Phase 0
16     **wait until** $(D1.h\_leader_p = id(p) \vee ((PH0, r_p, v)$ received);
17     **if** $((PH0, r_p, v)$ received) **then** $est1_p \leftarrow v$ **end if**;
18     broadcast$(PH0, r_p, est1_p)$;
19     // Phase 1
20     $sr_p \leftarrow 1$; $current\_labels_p \leftarrow D2.h\_labels_p$;
21     broadcast $(PH1, id(p), r_p, sr_p, current\_labels_p, est1_p)$;
22     **repeat**
23       **if** $((PH2, -, r_p, -, -, est2)$ received) **then**
24         $est2_p \leftarrow est2$; **exit** inner **repeat loop end if**;
25       **if** $((\exists (x, mset) \in D2.h\_quora_p) \wedge (\exists sr \in N) \wedge$
26          $(\exists$ set $M$ of messages $(PH1, -, r_p, sr, -, -))$, such that,
27          $(\forall (PH1, -, -, -, cl, -) \in M, x \in cl) \wedge$
28          $(mset = \{i : (PH1, i, -, -, -, -) \in M\}))$ **then**
29          **if** (all msgs in $M$ contain the same estimate $v$) **then**
30              $est2_p \leftarrow v$ **else** $est2_p \leftarrow \perp$ **end if**;
31          **exit** inner **repeat loop**;
32       **else if** $(current\_labels_p \neq D2.h\_labels_p) \vee$
33              $((PH1, -, r_p, sr, -, -)$ received with $sr > sr_p)$ **then**
34              $sr_p \leftarrow sr_p + 1$; $current\_labels_p \leftarrow D2.h\_labels_p$;
35              broadcast $(PH1, id(p), r_p, sr_p, current\_labels_p, est1_p)$
36            **end if**
37       **end if**
38     **end repeat**;
39     // Phase 2
40     $sr_p \leftarrow 1$; $current\_labels_p \leftarrow D2.h\_labels_p$;
41     broadcast $(PH2, id(p), r_p, sr_p, current\_labels_p, est2_p)$;
42     **repeat**
43       **if** $((COORD, -, r_p + 1, -)$ received) **then**
44          **exit** inner **repeat loop end if**;
45       **if** $((\exists (x, mset) \in D2.h\_quora_p) \wedge (\exists sr \in N) \wedge$
46          $(\exists$ set $M$ of messages $(PH2, -, r_p, sr, -, -))$, such that,
47          $(\forall (PH2, -, -, -, cl, -) \in M, x \in cl) \wedge$
48          $(mset = \{i : (PH2, i, -, -, -, -) \in M\}))$ **then**
49          **let** $rec_p$ = the set of estimates contained in $M$;
50          **if** $((rec_p = \{v\}) \wedge (v \neq \perp))$ **then**
51            broadcast $(DECIDE, v)$; **return**(v) **end if**;
52          **if** $((rec_p = \{v, \perp\}) \wedge (v \neq \perp))$ **then** $est1_p \leftarrow v$ **end if**;
53          **if** $(rec_p = \{\perp\})$ **then** skip **end if**;
54          **exit** inner **repeat loop**
55       **else if** $((current\_labels_p \neq D2.h\_labels_p) \vee$
56               $((PH2, -, r_p, sr, -, -)$ received with $sr > sr_p))$ **then**
57               $sr_p \leftarrow sr_p + 1$; $current\_labels_p \leftarrow D2.h\_labels_p$;
58               broadcast $(PH2, id(p), r_p, sr_p, current\_labels_p, est2_p)$
59            **end if**
60       **end if**
61     **end repeat**
62   **end repeat**;
63
64   **Task** T2
65    **upon reception** of $(DECIDE, v)$ **do**
66      broadcast $(DECIDE, v)$; **return**(v)
```

Figure 5. Consensus algorithm in $HAS[H\Omega, H\Sigma]$ (code for process p). It uses detectors $D1 \in H\Omega$ and $D2 \in H\Sigma$.

guarantees the liveness property for p. Then, there is a time in which $(x, m) \in D2.h\_quora_p$ and every correct process q in $S(x) \cap Correct$ has $x \in D2.h\_labels_q$. Note that, from Lines 32-36, every change in the variable $D2.h\_labels$ of a process creates a new sub-round, and that all processes broadcast their current value of $D2.h\_labels$ in each new sub-round. Therefore, eventually, p will receive messages $(PH1, -, r, sr, cl, -)$ from all these processes such that $x \in cl$. Hence, the condition of Lines 25-28 is satisfied, and p will exit the loop of Phase 1. The argument for the repeat loop of Phase 2 is verbatim. ■

Lemma 6. *No two processes decide different values in the same round.*

Proof: Let us assume that processes p_1 and p_2 decide values v_1 and v_2 in sub-rounds sr_1 and sr_2, respectively, of the same round r (in Line 51). Let (x_1, m_1) and M_1 be the pair in $D2.h\_quora_{p_1}$ and the set of messages that satisfy the condition of Lines 45-48 for p_1. Since for each message $(PH2, -, r, sr_1, cl, -) \in M_1$, it holds that $x_1 \in cl$, if Q_1 is the set of senders of the messages in M_1, we have that $Q_1 \subseteq S(x_1)$. Additionally, $m_1 = \{i : (PH2, i, -, -, -, -) \in M_1\} = I(Q_1)$. We can define (x_2, m_2) and M_2 analogously for p_2. Then, from the Safety Property of $H\Sigma$, $Q_1 \cap Q_2 \neq \emptyset$. Let $p_l \in Q_1 \cap Q_2$. Then, process p_l must have broadcast messages $(PH2, id(p_l), r, sr_1, -, v_1)$ and $(PH2, id(p_l), r, sr_2, -, v_2)$ (Lines 41 and 58). Since the estimate $est2_{p_l}$ of p_l does not change between sub-rounds (inner repeat loop, Lines 42-61), it must hold that $v_1 = v_2$. From the condition of Line 51, $rec_{p_1} = \{v_1\}$ in sub-round sr_1 and $rec_{p_2} = \{v_2\}$ in sub-round sr_2, and both processes decide the same value. Hence, no two processes decide different values in the same round. ■

Theorem 7. *The algorithm of Figure 5 solves consensus in* $HAS[H\Omega, H\Sigma]$.

Proof: The proof of this theorem is similar to the proof of Theorem 5 of [5] (full version of [4]), with the following changes. Observe that the Leaders' Coordination Phase and Phase 0 of the algorithms in Figures 4 and 5 are the same. Hence, Lemmas 3 and 4 also apply to the algorithm of Figure 5. Then, the termination property can be proven in a similar way as in [5] (Lemmas 1 and 2), but using those two Lemmas 3 and 4 together with Lemma 5. The proof of the agreement property is also similar to Lemma 3 of [5] but using Lemma 6. ■

The algorithm of Figure 5 can be easily transformed into an algorithm that solves consensus in $AAS[A\Omega, H\Sigma]$ (an anonymous system with detectors $A\Omega$ and $H\Sigma$). For that, given a failure detector $D3 \in A\Omega$, it is enough to remove the Leaders' Coordination Phase, and in Phase 0 to replace $(D1.h\_leader_p = id(p))$ by $(D3.a\_leader_p)$. The resulting Phase 0 is the same as Phase 1 in the algorithm of Figure 3 of [4], and has the same properties.

References

[1] D. Angluin, Local and global properties in networks of processors (extended abstract). In *STOC*, pages 82-93. ACM Press, 1980.

[2] H. Attiya, M. Snir, and M. Warmuth, Computing on an anonymous ring. *J. ACM*, 35(4):845–875, 1988.

[3] F. Bonnet and M. Raynal, The price of anonymity: Optimal consensus despite asynchrony, crash and anonymity. *DISC*, volume 5805 of *LNCS*, pp. 341-355. Springer, 2009.

[4] F. Bonnet and M. Raynal, Anonymous asynchronous systems: The case of failure detectors. *DISC*, volume 6343 of *LNCS*, pages 206-220. Springer, 2010.

[5] F. Bonnet and M. Raynal, Anonymous asynchronous systems: The case of failure detectors. Technical Report PI 1945, IRISA, Rennes, France, January 2010.

[6] F. Bonnet and M. Raynal, Consensus in anonymous distributed systems: Is there a weakest failure detector? In *AINA*, pp. 206-213, IEEE Computer Society, 2010.

[7] Z. Bouzid, P. Sutra, and C. Travers, Anonymous agreement: the janus algorithm. To appear, *OPODIS*, 2011.

[8] T.D. Chandra, V. Hadzilacos, and S. Toueg, The weakest failure detector for solving consensus. *J. ACM*, 43(4):685-722, 1996.

[9] T.D. Chandra and S. Toueg, Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225-267, 1996.

[10] C. Delporte-Gallet, H. Fauconnier, and R. Guerraoui, A realistic look at failure detectors. In *DSN*, pages 345-353. IEEE Computer Society, 2002.

[11] C. Delporte-Gallet, H. Fauconnier, and R. Guerraoui, Tight failure detection bounds on atomic object implementations. *J. ACM*, 57(4), 2010.

[12] C. Delporte-Gallet, H. Fauconnier, R. Guerraoui , A.-M. Kermarrec, E. Ruppert and H. Tran-The, Byzantine agreement with homonymous. In *PODC*, 2011.

[13] C. Delporte-Gallet, H. Fauconnier and A. Tielmann, Fault-toleranr consensus in Unknown and anonymous networks. In *ICDCS*, pages 368-375, IEEE Computer Society, 2009.

[14] M. Fischer, N. Lynch and M. Paterson, Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374-382, 1985.

[15] F. Greve and S. Tixeuil, Knowledge connectivity vs synchrony requirements for fault-tolerant agreement in unknow networks. In *DSN*, pp. 82-91, IEEE Computer Society, 2007.

[16] E. Jiménez, S. Arévalo, and A. Fernández, Implementing unreliable failure detectors with unknown membership. *Inf. Process. Lett.*, 100(2):60–63, 2006.

[17] N. Lynch, *Distributed Algorithms*. Morgan Kaufmann Pub., San Francisco (CA), 1996.

[18] M. Raynal, Failure detectors for asynchronous distributed systems: an introduction. In *Wiley Encyclopedia of Computer Science and Engineering*, volume 2, pages 1181-1191. 2009.

[19] M. Raynal, *Communication and Agreement Abstractions for Fault-Tolerant Asynchronous Distributed Systems*. 250 pages, Morgan & Claypool Publishers, 2010.

[20] M. Yamashita and T. Kameda, Computing on anonymous networks: Part i-characterizing the solvable cases. *IEEE Trans. Parallel Distributed Systems*, 7(1):69-89, 1996.

[21] P. Zielinski, Anti-omega: the weakest failure detector for set agreement. *PODC*, pp. 55-64. ACM Press, , 2008.

```
1  Init
2    $h\_labels_p \leftarrow \{s : (s \subseteq I(\Pi)) \wedge (id(p) \in s)\}$;
3    $h\_quora_p \leftarrow \emptyset$;
4    repeat forever
5      $q \leftarrow D.trusted_p$;
6      $h\_quora_p \leftarrow h\_quora_p \cup \{(q, q)\}$;
7    end repeat;
```

Figure 6. Algorithm to transform $D \in \Sigma$ to $H\Sigma$ with initial knowledge of membership (code for process p).

```
1  Init
2    $h\_labels_p \leftarrow \emptyset$;
3    $h\_quora_p \leftarrow \emptyset$;
4    $mship_p \leftarrow \emptyset$;
5    start tasks T1 and T2;
6  Task T1
7    repeat forever
8      broadcast $(IDENT, id(p))$;
9      $q \leftarrow D.trusted_p$;
10     $h\_quora_p \leftarrow h\_quora_p \cup \{(q, q)\}$;
11   end repeat;
12
13 Task T2
14   upon reception of $(IDENT, i)$ do
15     $mship_p \leftarrow mship_p \cup \{i\}$
16     $h\_labels_p \leftarrow \{s : (s \subseteq mship_p) \wedge (id(p) \in s)\}$;
```

Figure 7. Algorithm to transform $D \in \Sigma$ to $H\Sigma$ without initial knowledge of membership (code for process p).

APPENDIX

A. Reductions between Failure Detectors

1) From Σ to $H\Sigma$: We prove that, if identifiers are unique, a detector of class $H\Sigma$ can be obtained from any detector D of class Σ.

Theorem 8. *A failure detector of class $H\Sigma$ can be obtained from any detector D of class Σ in a system with unique identifiers, under either one of the following conditions:*

1) *without any communication if every process initially knows the membership $I(\Pi)$, or*
2) *in system $AS[\Sigma]$ (the membership does not need to be known initially).*

Proof: Let $D.trusted_p$ be the variable of Σ failure detector D at process p. Figures 6 and 7 present the algorithms to transform D into a failure detector of class $H\Sigma$ in Cases 1 and 2, respectively. In both cases, at each process p initially $h\_quora_p \leftarrow \emptyset$, and infinitely often this variable is updated with the following sentences: $q \leftarrow D.trusted_p$, and $h\_quora_p \leftarrow h\_quora_p \cup \{(q, q)\}$. In Case 1, initially every process p sets $h\_labels_p \leftarrow \{s : (s \subseteq I(\Pi)) \wedge (id(p) \in s)\}$ and it never changes it in the run. In Case 2, every process p initially sets $h\_labels_p \leftarrow \emptyset$, and repeatedly broadcasts a message $IDENT(id(p))$. Process p also has a variable $mship_p$ initially set to $mship_p \leftarrow \emptyset$. After receiving a message $IDENT(i)$, process p updates $mship_p \leftarrow mship_p \cup \{i\}$, and $h\_labels_p \leftarrow \{s : (s \subseteq mship_p) \wedge (id(p) \in s)\}$.

We prove now the properties of $H\Sigma$:

- Validity. Since h_quora_p is a set, and the elements included in it are of the form (q, q) (see Line 5 in Figure 6, and Line 10 in Figure 7) there can not be two pairs with the same label.
- Monotonicity. The monotonicity of h_labels_p in Figure 6 is obvious because it is initialized in Line 2 and never changes. With respect to Figure 7, h_labels_p is initially empty, and it is related with the set $mship_p$, such that if $mship_p$ grows then h_labels_p either grows or remains the same. Hence h_labels_p never decreases because $mship_p$ never decreases (see Line 15 in Figure 7). The monotonicity of h_quora_p in Figures 6 and 7 follows from the fact that h_quora_p is initially empty, and any element (q, q) included in it is never removed.
- Liveness. Consider any correct process p. In Figure 7, eventually, $Correct \subseteq mship_p$ permanently (from the exchange of $IDENT$ messages and Line 15 of Figure 7). Then, in both algorithms eventually $\{s : (s \subseteq I(Correct)) \wedge (id(p) \in s)\} \subseteq h\_labels_p$ permanently (from Line 2 in Figure 6, and Line 16 in Figure 7). Hence, there is a time τ after which, for every set $s \subseteq I(Correct)$, $I(S(s)) = s$ and $S(s) \subseteq Correct$.
  The Liveness property of Σ guarantees that, at some time $\tau' \geq \tau$, the variable q is assigned a set s that contains only correct processes and (s, s) will be included in h_quora_p after that. Therefore, there is a time after which h_quora_p contains (s, s) permanently (from monotonicity). Since $s \subseteq I(S(s) \cap Correct) = I(S(s)) = s$, the property follows.
- Safety. Consider two pairs $(x_1, m_1) \in h\_quora_{p_1}^{\tau_1}$ and $(x_2, m_2) \in h\_quora_{p_2}^{\tau_2}$, for any $p_1, p_2 \in \Pi$ and any $\tau_1, \tau_2 \in N$. From the management of the h_quora variables (Lines 3, 5, and 6 in Figure 6, and Lines 3, 9, and 10 in Figure 7), we have that m_1 and m_2 are values taken from $D.trusted_{p_1}$ and $D.trusted_{p_2}$, respectively. Hence, the sets m_1 and m_2 must intersect from the Safety property of the Σ failure detector D. Then, if $I(Q_1) = m_1$ and $I(Q_2) = m_2$, given that we are in a system with unique identifiers, Q_1 and Q_2 must intersect.

■

2) From $H\Sigma$ to Σ: We define now a new class of failure detector that will be used for reductions between the above failure detector classes. While the service provided by this detector has been already used [21], [4], it was never formally defined. The new failure detector class, denoted Ξ, will only be defined for systems with unique identifiers, i.e., non homonymous.

Definition 1. *A failure detector of class Ξ provides each process $p \in \Pi$, in a system with unique process identifiers, with a variable $alive_p$ which contains a (sorted) list of process identifiers. Any failure detector of class Ξ must*

```
1  Init
2    start Tasks T1 and T2;
3  Task T1
4    repeat forever
5      broadcast $(LABELS, id(p), D.h\_labels_p)$;
6      if $\exists(x,m) \in D.h\_quora_p : (idents_p[x]$ has been created$) \wedge (m \subseteq idents_p[x])$ then
7        let $candidates_p = \{m : ((x,m) \in D.h\_quora_p) \wedge (idents_p[x]$ has been created$) \wedge (m \subseteq idents_p[x])\}$;
8        $trusted_p \leftarrow$ any $m \in candidates_p$ with smallest $\max_{i \in m} rank(i, X.alive_p)$;
9      end if;
10   end repeat;
11
12 Task T2
13   upon reception of $(LABELS, i, \ell)$ do
14     foreach $x \in \ell$ do
15       if $idents_p[x]$ has not been created then create $idents_p[x] \leftarrow \emptyset$ end if;
16       $idents_p[x] \leftarrow idents_p[x] \cup \{i\}$;
17     end foreach;
```

Figure 9. Algorithm to transform $D \in H\Sigma$ to Σ in a system with unique identifiers, but without initial knowledge of membership (code for process p). The algorithm uses a failure detector X of class Ξ.

```
Init
  $alive_p \leftarrow$ empty list;
  start Tasks T1 and T2;
Task T1
  repeat forever
    broadcast $(ALIVE, id(p))$;
  end repeat;

Task T2
  upon reception of $(ALIVE, i)$ do
    if $i \in alive_p$ then move $i$ to the first position of $alive_p$
    else insert $i$ in the first position of $alive_p$
    end if;
```

Figure 8. Algorithm to implement a failure detector of class Ξ without initial knowledge of membership in AS[$\emptyset$] (code for process p).

satisfy the following property:

- *Liveness. Eventually, the identifiers of the correct processes are permanently in the first positions of $alive_p$. More formally, let $rank(i, alive_p^\tau)$ denote the position (starting from 1) of process identifier i in $alive_p^\tau$ (with $rank(i, alive_p^\tau) = \infty$ if $i \notin alive_p^\tau$). Then, $\forall p \in Correct, \exists \tau \in N : \forall \tau' \geq \tau, \forall q \in Correct, rank(id(q), alive)_p^{\tau'} \leq |Correct|$.*

Observe that the position of the same identifier can be different at different processes, and can vary over time in the same process. From the algorithm of Figure 8, we obtain the following lemma.

Lemma 7. *A failure detector of class Ξ can be implemented in AS[$\emptyset$] (an asynchronous system with unique identifiers), even when the membership is not known initially.*

Proof: For each process $q \in Correct$, eventually some message $ALIVE(id(q))$ will be received at each process $p \in Correct$. Then $id(q)$ will be included in $alive_p$ and never removed after that. Given any faulty process r, p will stop receiving messages from r by some time τ. Then, after τ process p will never receive a message $ALIVE(id(r))$ and $id(r)$ will never be moved to (inserted in) the first position of $alive_p$. However, after τ, eventually p will receive messages $ALIVE(id(q))$ from each process $q \in Correct$, and each identifier $id(q)$ will be moved to (or inserted in) the first position of $alive_p$. Then, there is some time $\tau' > \tau$ such that, at all times $\tau'' > \tau'$, $rank(id(q), alive_p^{\tau''}) < rank(id(r), alive_p^{\tau''})$. Since this holds for all $p, q \in Correct$ and all $r \notin Correct$, the claim follows. ■

We now show, using the algorithm of Figure 9, that Σ can be obtained from $H\Sigma$ without initial knowledge of the membership.

Theorem 9. *A failure detector of class Σ can be obtained from any detector D of class $H\Sigma$ in AS[$H\Sigma$] (an asynchronous system with unique identifiers), even when the membership is not known initially.*

Proof: From Lemma 7, we can have a failure detector of class Ξ in an asynchronous system. The logic of the algorithm of Figure 9 is somewhat similar to that of the algorithm in Figure 2 in [4]. The condition in Line 6 guarantees that the variable $trusted_p$ is assigned a set of identifiers m only if (x, m) is in h_quora_p, and every process q whose identifier is in m has x in its set h_labels_q (from the management of the sets $idents_p$). Combining this condition with the safety property of $H\Sigma$ we guarantee the safety property of Σ. The liveness property of Σ holds from the liveness property of $H\Sigma$, the choice of m done in Line 8, and the properties of the failure detector class Ξ as follows. If $p \in Correct$, from the liveness of $H\Sigma$, eventually every time Line 8 is executed, there is some $m \in candidates_p$ with only correct processes. If the failure detector X of class Ξ has already all the correct processes in the lowest ranks of $X.alive_p$ (which eventually happens from its liveness property), then any set m in $candidates_p$, whose largest rank in $X.alive_p$ is minimal, contains only correct processes (which yields the liveness of Σ). ■

Theorem 1 *Failure detector classes Σ, $H\Sigma$, and $A\Sigma$ are equivalent in $AS[\emptyset]$. Furthermore, the transformation between Σ and $H\Sigma$ do not require initial knowledge of the membership.*

Proof of Theorem 1 From Theorems 8 and 9 we have that Σ and $H\Sigma$ are equivalent. The equivalence between Σ and $A\Sigma$ was shown in [4].

3) From $A\Sigma$ to $H\Sigma$: We show now how to obtain a failure detector of class $H\Sigma$ from a detector of class $A\Sigma$.

Theorem 2 *Class $H\Sigma$ can be obtained from class $A\Sigma$ in $AAS[\emptyset]$ without communication.*

Proof of Theorem 2 Let D be a detector of class $A\Sigma$. The transformation can be done as follows. Let $\perp$ be the "default" identifier. Let us denote with $\perp^r$ a multiset of r identifiers $\perp$. Each process p periodically does as follows. For each pair $(x, y) \in D.a\_sigma_p$, the label x is included in h_labels_p and the pair $(x, \perp^y)$ is included in h_quora_p (replacing any pair $(x, -)$ that h_quora_p may contain). The properties of $H\Sigma$ follow trivially from the properties of $A\Sigma$.

4) From $\overline{AP}$ to $\diamond H\overline{P}$ and $H\Sigma$: We show here how failure detectors of the classes $\diamond H\overline{P}$ and $H\Sigma$ can be obtained for a failure detector of class $\overline{AP}$ without communication.

Lemma 8. *A failure detector of class $\diamond H\overline{P}$ can be obtained from any detector D of class $\overline{AP}$ in $AAS[\emptyset]$ (an anonymous asynchronous system) without communication.*

Proof: The transformation can be done as follows. Let $\perp$ be the "default" identifier. Each process p periodically updates $h\_trusted_p$ to a multiset of $D.anap_p$ identifiers $\perp$. The liveness property of D guarantees the liveness property of $\diamond H\overline{P}$. ■

Lemma 9. *A failure detector of class $H\Sigma$ can be obtained from any detector D of class $\overline{AP}$ in $AAS[\emptyset]$ (an anonymous asynchronous system) without communication.*

Proof: The transformation can be done as follows. Let $\perp$ be the "default" identifier. Let us denote with $\perp^r$ a multiset of r identifiers $\perp$. Each process p periodically does as follows. After obtaining a value y from $D.anap_p$, the label $\perp^y$ is included in h_labels_p and the pair $(\perp^y, \perp^y)$ is included in h_quora_p. The Validity and Monotonicity of $H\Sigma$ hold trivially. Liveness follows since, from the safety of $\overline{AP}$, only correct processes see an output of $D.anap = c = |Correct|$, and from the liveness property all of them do it. Then, every correct process p eventually inserts $\perp^c$ in h_labels_p and $(\perp^c, \perp^c)$ in h_quora_p, and only those processes. Safety of $H\Sigma$ comes from the safety property of $\overline{AP}$: if, for any y and y' with $y \geq y'$, $|S(\perp^y)| = y$ and $|S(\perp^{y'})| = y'$ (none can be larger), then $S(\perp^y) \subseteq S(\perp^{y'})$. ■

Theorem 3 *Classes $\diamond H\overline{P}$ and $H\Sigma$ can be obtained from class $\overline{AP}$ in $AAS[\emptyset]$ without communication.*

Proof of Theorem 3 The proof of Theorem 3 follows from the two previous lemmas.